Lean Type Theory

Riccardo Brasca

Atelier Lean 2023

May 3rd 2023

In Lean (almost) *everything* is a term

In Lean (almost) *everything* is a term and has a type

In Lean (almost) *everything* is a term and has a type

$$(0 : \mathbb{N})$$

In Lean (almost) *everything* is a term and has a type

$$(0 : \mathbb{N}) \quad (\pi : \mathbb{R}) \quad \ldots$$

In Lean (almost) *everything* is a term and has a type

$$(0 : \mathbb{N}) \quad (\pi : \mathbb{R}) \quad \dots$$

$\mathbb{N}$ and $\mathbb{R}$ are *types*

In Lean (almost) *everything* is a term and has a type

$$(0 : \mathbb{N}) \quad (\pi : \mathbb{R}) \quad \ldots$$

$\mathbb{N}$ and $\mathbb{R}$ are *types*, but also terms, so they have their own type:

$$(\mathbb{N} : \mathrm{Type}) \quad (\mathbb{R} : \mathrm{Type})$$

In Lean (almost) *everything* is a term and has a type

$$(0 : \mathbb{N}) \quad (\pi : \mathbb{R}) \quad \ldots$$

$\mathbb{N}$ and $\mathbb{R}$ are *types*, but also terms, so they have their own type:

$$(\mathbb{N} : \mathrm{Type}) \quad (\mathbb{R} : \mathrm{Type})$$

Type is also a term!

In Lean (almost) *everything* is a term and has a type

$$(0 : \mathbb{N}) \quad (\pi : \mathbb{R}) \quad \dots$$

$\mathbb{N}$ and $\mathbb{R}$ are *types*, but also terms, so they have their own type:

$$(\mathbb{N} : \mathrm{Type}) \quad (\mathbb{R} : \mathrm{Type})$$

Type is also a term! It has type Type 1

In Lean (almost) *everything* is a term and has a type

$$(0 : \mathbb{N}) \quad (\pi : \mathbb{R}) \quad \ldots$$

$\mathbb{N}$ and $\mathbb{R}$ are *types*, but also terms, so they have their own type:

$$(\mathbb{N} : \mathrm{Type}) \quad (\mathbb{R} : \mathrm{Type})$$

Type is also a term! It has type Type 1

Type and Type 1 are *universes*

Lean has a countable hierarchy of universes

Lean has a countable hierarchy of universes

$$\text{Type } 0$$

Lean has a countable hierarchy of universes

$$\text{Type } 0 = \text{Type}$$

Lean has a countable hierarchy of universes

$$\text{Type } 0 = \text{Type}$$

$$\text{Type } 1$$

Lean has a countable hierarchy of universes

$$\text{Type } 0 = \text{Type}$$

$$\text{Type } 1$$

$$\text{Type } 2$$

Lean has a countable hierarchy of universes

$$\text{Type } 0 = \text{Type}$$

$$\text{Type } 1$$

$$\text{Type } 2$$

And so on.

Lean has a countable hierarchy of universes

$$\text{Type } 0 = \text{Type}$$

$$\text{Type } 1$$

$$\text{Type } 2$$

And so on. In general

$$\text{Type } n$$

Lean has a countable hierarchy of universes

$$\text{Type } 0 = \text{Type}$$

$$\text{Type } 1$$

$$\text{Type } 2$$

And so on. In general

$$\text{Type } n : \text{Type } n+1$$

Lean has a countable hierarchy of universes

$$\mathrm{Type}\ 0 = \mathrm{Type}$$

$$\mathrm{Type}\ 1$$

$$\mathrm{Type}\ 2$$

And so on. In general

$$\mathrm{Type}\ n : \mathrm{Type}\ n + 1$$

At the very bottom there is a special universe: $\mathrm{Prop} : \mathrm{Type}$

Everything has its own type.

Everything has its own type. If P is a mathematical statement, then $P : \mathrm{Prop}$

Everything has its own type. If P is a mathematical statement, then $P : \mathrm{Prop}$

What about proofs?

Everything has its own type. If P is a mathematical statement, then $P : \mathrm{Prop}$

What about proofs? A proof of P is *a term of type* P!

Everything has its own type. If P is a mathematical statement, then $P : \mathrm{Prop}$

What about proofs? A proof of P is *a term of type* P!

```
lemma easy : ∀ n, n + 0 = n :=
begin
  intro n,
  refl,
end
```

Everything has its own type. If P is a mathematical statement, then $P : \mathrm{Prop}$

What about proofs? A proof of P is *a term of type P*!

```
lemma easy : ∀ n, n + 0 = n :=
begin
  intro n,
  refl,
end
```

The type of easy is the statement $\forall n, n + 0 = n$

Everything has its own type. If P is a mathematical statement, then $P : \mathrm{Prop}$

What about proofs? A proof of P is *a term of type* P!

```
lemma easy : ∀ n, n + 0 = n :=
begin
  intro n,
  refl,
end
```

The type of easy is the statement $\forall n, n + 0 = n$: to specify this term we had to prove the property

Everything has its own type. If P is a mathematical statement, then $P : \mathrm{Prop}$

What about proofs? A proof of P is *a term of type P*!

```
lemma easy : ∀ n, n + 0 = n :=
begin
  intro n,
  refl,
end
```

The type of easy is the statement $\forall n, n + 0 = n$: to specify this term we had to prove the property
Proposition are special because of *proof irrelevance*.

How to build new types out of old ones?

How to build new types out of old ones? Two fundamental constructions

How to build new types out of old ones? Two fundamental constructions: Π-types and Σ-types

How to build new types out of old ones? Two fundamental constructions: Π-types and Σ-types

- If A and B are types we have the type of functions $A \to B$.

How to build new types out of old ones? Two fundamental constructions: Π-types and Σ-types

- If A and B are types we have the type of functions $A \to B$. More generally we have the *dependent* Π-*type*: if f is a term of this type then the type of $f\ a$ depends on $(a : A)$

How to build new types out of old ones? Two fundamental constructions: Π-types and Σ-types

- If A and B are types we have the type of functions $A \to B$. More generally we have the *dependent* Π*-type*: if f is a term of this type then the type of $f\ a$ depends on $(a : A)$
- If A and B are types we have the Cartesian product $A \times B$.

How to build new types out of old ones? Two fundamental constructions: Π-types and Σ-types

- If A and B are types we have the type of functions $A \to B$. More generally we have the *dependent* Π-*type*: if f is a term of this type then the type of $f\ a$ depends on $(a : A)$
- If A and B are types we have the Cartesian product $A \times B$. More generally we have the *dependent* Σ-*type*: if (a, b) is a term of this type then the type of b depends on $(a : A)$

How to build new types out of old ones? Two fundamental constructions: Π-types and Σ-types

- If A and B are types we have the type of functions $A \to B$. More generally we have the *dependent* Π*-type*: if f is a term of this type then the type of $f\ a$ depends on $(a : A)$
- If A and B are types we have the Cartesian product $A \times B$. More generally we have the *dependent* Σ*-type*: if (a, b) is a term of this type then the type of b depends on $(a : A)$

These types come with certains axioms that allow to build terms and use them

If P and Q are proposition, to build a function $f : P \to Q$ we need to specify a term $f\ p$ of type Q for all terms p of type P

If P and Q are proposition, to build a function $f : P \to Q$ we need to specify a term $f\ p$ of type Q for all terms p of type P

This is the same as giving a proof of Q having a proof of P

If P and Q are proposition, to build a function $f : P \to Q$ we need to specify a term $f\ p$ of type Q for all terms p of type P

This is the same as giving a proof of Q having a proof of P, so proving that P implies Q

If P and Q are proposition, to build a function $f : P \to Q$ we need to specify a term $f\ p$ of type Q for all terms p of type P

This is the same as giving a proof of Q having a proof of P, so proving that P implies Q

$$P \to Q \text{ is the same as } P \Rightarrow Q$$

If P and Q are proposition, to build a function $f : P \to Q$ we need to specify a term $f\ p$ of type Q for all terms p of type P

This is the same as giving a proof of Q having a proof of P, so proving that P implies Q

$$P \to Q \text{ is the same as } P \Rightarrow Q$$

Proofs are functions that take a list of assumptions

$$(h_1 : P_1) \dots (h_n : P_n)$$

and produce a proof a proposition P

If P and Q are proposition, to build a function $f : P \to Q$ we need to specify a term $f\ p$ of type Q for all terms p of type P

This is the same as giving a proof of Q having a proof of P, so proving that P implies Q

$$P \to Q \text{ is the same as } P \Rightarrow Q$$

Proofs are functions that take a list of assumptions

$$(h_1 : P_1) \dots (h_n : P_n)$$

and produce a proof a proposition P

By proof irrelevance it doesn't matter which proof of P_i we suppose to have.

If P and Q are proposition, to build a function $f : P \to Q$ we need to specify a term $f\ p$ of type Q for all terms p of type P

This is the same as giving a proof of Q having a proof of P, so proving that P implies Q

$$P \to Q \text{ is the same as } P \Rightarrow Q$$

Proofs are functions that take a list of assumptions

$$(h_1 : P_1) \dots (h_n : P_n)$$

and produce a proof a proposition P

By proof irrelevance it doesn't matter which proof of P_i we suppose to have. $(h_i : P_i)$ means that P_i is provable

How to build $\mathbb{N}$?

How to build $\mathbb{N}$? It is an *inductive type*!

An inductive type T has

How to build $\mathbb{N}$? It is an *inductive type*!

An inductive type T has:

- finitely many constructors.

How to build $\mathbb{N}$? It is an *inductive type*!

An inductive type T has:

- finitely many constructors. They allow to build terms of type T.

How to build $\mathbb{N}$? It is an *inductive type*!

An inductive type T has:

- finitely many constructors. They allow to build terms of type T. T itself can appear as an argument of the constructor (with some conditions).

How to build $\mathbb{N}$? It is an *inductive type*!

An inductive type T has:

- finitely many constructors. They allow to build terms of type T. T itself can appear as an argument of the constructor (with some conditions). They're used to define functions *in* T

How to build $\mathbb{N}$? It is an *inductive type*!

An inductive type T has:

- finitely many constructors. They allow to build terms of type T. T itself can appear as an argument of the constructor (with some conditions). They're used to define functions *in* T

- An induction principle that combines recursion and mathematical induction.

How to build $\mathbb{N}$? It is an *inductive type*!

An inductive type T has:

- finitely many constructors. They allow to build terms of type T. T itself can appear as an argument of the constructor (with some conditions). They're used to define functions *in* T
- An induction principle that combines recursion and mathematical induction. It allows to define functions *out* of T

How to build $\mathbb{N}$? It is an *inductive type*!

An inductive type T has:

- finitely many constructors. They allow to build terms of type T. T itself can appear as an argument of the constructor (with some conditions). They're used to define functions *in* T

- An induction principle that combines recursion and mathematical induction. It allows to define functions *out* of T (and prove theorems about T)

How to build $\mathbb{N}$? It is an *inductive type*!

An inductive type T has:

- finitely many constructors. They allow to build terms of type T. T itself can appear as an argument of the constructor (with some conditions). They're used to define functions *in* T

- An induction principle that combines recursion and mathematical induction. It allows to define functions *out* of T (and prove theorems about T)

We allow inductive propositions

How to build $\mathbb{N}$? It is an *inductive type*!

An inductive type T has:

- finitely many constructors. They allow to build terms of type T. T itself can appear as an argument of the constructor (with some conditions). They're used to define functions *in* T

- An induction principle that combines recursion and mathematical induction. It allows to define functions *out* of T (and prove theorems about T)

We allow inductive propositions
Using the induction principle we can prove that the constructors are injective

Examples of inductive types:

Examples of inductive types:

- $\mathbb{N}$

Examples of inductive types:

- $\mathbb{N}$
- $\mathrm{true} : \mathrm{Prop}$

Examples of inductive types:

- $\mathbb{N}$
- $\mathrm{true} : \mathrm{Prop}$ and $\mathrm{false} : \mathrm{Prop}$

Examples of inductive types:

- $\mathbb{N}$
- $\mathrm{true} : \mathrm{Prop}$ and $\mathrm{false} : \mathrm{Prop}$
- $\exists$, $\wedge$ and $\vee$

Examples of inductive types:

- $\mathbb{N}$
- $\mathrm{true} : \mathrm{Prop}$ and $\mathrm{false} : \mathrm{Prop}$
- $\exists$, $\wedge$ and $\vee$
- Equality is *defined* in Lean, and it is an inductive proposition!

Examples of inductive types:

- $\mathbb{N}$
- $\mathrm{true} : \mathrm{Prop}$ and $\mathrm{false} : \mathrm{Prop}$
- $\exists$, $\wedge$ and $\vee$
- Equality is *defined* in Lean, and it is an inductive proposition!

One can somehow do induction on equality

Mathlib's axioms

Mathlib adds three axioms to the theory

Mathlib's axioms

Mathlib adds three axioms to the theory

- Propositional extensionality

Mathlib's axioms

Mathlib adds three axioms to the theory

- Propositional extensionality
- Construction of quotient types

Mathlib's axioms

Mathlib adds three axioms to the theory

- Propositional extensionality
- Construction of quotient types
- The axioms of choice

Mathlib's axioms

Mathlib adds three axioms to the theory

- Propositional extensionality
- Construction of quotient types
- The axioms of choice

It is known to be consistent to Zermelo Fraenkel with choice and existence n inaccessible cardinals for all n

In particular the following are theorems

In particular the following are theorems

- Functional extensionality

In particular the following are theorems

- Functional extensionality
- Excluded middle

In particular the following are theorems

- Functional extensionality
- Excluded middle
- If $P : \mathrm{Prop}$ then $P = \mathrm{true}$ of $P = \mathrm{false}$